Long Distance Q-Resolution with Dependency Schemes

Tomáš Peitl, Friedrich Slivovsky, and Stefan Szeider(✉)

Algorithms and Complexity Group, TU Wien, Vienna, Austria
{peitl,fslivovsky,sz}@ac.tuwien.ac.at

Abstract. Resolution proof systems for quantified Boolean formulas (QBFs) provide a formal model for studying the limitations of state-of-the-art search-based QBF solvers, which use these systems to generate proofs. In this paper, we define a new proof system that combines two such proof systems: Q-resolution with generalized universal reduction according to a dependency scheme and long distance Q-resolution. We show that the resulting proof system is sound for the reflexive resolution-path dependency scheme—in fact, we prove that it admits strategy extraction in polynomial time. As a special case, we obtain soundness and polynomial-time strategy extraction for long distance Q-resolution with universal reduction according to the standard dependency scheme. We report on experiments with a configuration of DepQBF that generates proofs in this system.

1 Introduction

Quantified Boolean Formulas (QBFs) offer succinct encodings for problems from domains such as formal verification, synthesis, and planning [5,10,13,27,33,37]. Although the combination of (more verbose) propositional encodings with SAT solvers is still the state-of-the-art approach to many of these problems, QBF solvers are gaining ground. An arsenal of new techniques has been introduced over the past few years [8,9,11,18,19,21,23,26,29,30,32], and these advances in solver technology have been accompanied by the development of a better understanding of the underlying QBF proof systems and their limitations [4,6,7,15,22,24,36].

Search-based solvers based on the QDPLL algorithm [12] represent one of the principal state-of-the-art approaches in QBF solving. Akin to modern SAT solvers, these solvers rely on successive variable assignments in combination with fast constraint propagation and learning. Unlike SAT solvers, however, search-based QBF solvers are constrained by the variable dependencies induced by the quantifier prefix[1]: while SAT solvers can assign variables in any order, search-based QBF solvers can only assign variables from the leftmost quantifier block

This research was partially supported by FWF grants P27721 and W1255-N23.

[1] We consider QBFs in prenex normal form.

N. Creignou and D. Le Berre (Eds.): SAT 2016, LNCS 9710, pp. 500–518, 2016.
DOI: 10.1007/978-3-319-40970-2_31

that contains unassigned variables, since the assignment of a variable further to the right might depend on the variable assignment to this block. In the most extreme case, this forces solvers into a fixed order of variable assignments, rendering decision variable heuristics ineffective.

The search-based solver DepQBF uses dependency schemes to partially bypass this restriction [8,28]. Dependency schemes can sometimes identify pairs of variables as independent, allowing the solver to assign them in any order. This gives decision heuristics more freedom and results in increased performance [8].

While this provides a strong motivation to use dependency schemes, their integration with QDPLL poses challenges of its own. Soundness of the proof system underlying QDPLL with the standard dependency scheme as implemented in DepQBF was shown only recently [36], and combining other state-of-the-art techniques with dependency schemes is often highly nontrivial. In this paper, we focus on two such issues:

(a) Long distance Q-resolution permits the derivation of tautological clauses in certain cases [2,39,40]. This system can be used in constraint learning as an alternative to Q-resolution, leading to fewer backtracks during search and, sometimes, reduced runtime [16]. In addition, clause learning based on long distance Q-resolution is substantially easier to implement. Currently, however, DepQBF does not permit learning based on long distance Q-resolution in conjunction with dependency schemes, as the resulting proof system is not known to be sound.
(b) For applications in verification and synthesis, it is not enough for solvers to decide whether an input QBF is true or false—they also have to generate a certificate. Such certificates can be efficiently constructed from Q-resolution [2] and even long distance Q-resolution proofs [3]. However, it is not clear whether this is possible for proofs generated by DepQBF with the standard dependency scheme, and proof generation with the standard dependency scheme is disabled by default.

We address (a) by showing that long distance Q-resolution combined with universal reduction according to the reflexive resolution-path dependency scheme [36] is sound. In fact, we prove that this proof system allows for certificate extraction in polynomial time, thus resolving (b). In particular, these results hold for the standard dependency scheme, which is weaker than the reflexive resolution-path dependency scheme.

Our proof of this result relies on an interpretation of Q-resolution refutations as winning strategies for the universal player in the evaluation game [20]. Defining LDQ(D) as the proof system consisting of long distance Q-resolution with universal reduction according to a dependency scheme D, we identify a natural property of a dependency scheme D that not only allows for the interpretation of an LDQ(D)-refutation as a winning strategy for the universal player, but even implies certificate extraction in time $O(|\mathcal{P}| \cdot n)$ from an LDQ(D)-refutation $\mathcal{P}$ of a QBF with n variables. We then show that the reflexive resolution path dependency scheme in fact has this property.

One of our motivations for studying the combination of long distance Q-resolution and dependency schemes is that all of the required logic is already implemented in DepQBF—it is simply that, by default, these features cannot be enabled at the same time because it is unclear whether the resulting solver configuration is sound. To complement our theoretical results, we bypassed these checks and performed experiments with DepQBF and constraint learning based on LDQ(D) with the standard dependency scheme. Our experiments show that performance with this type of learning is on par with and—in some cases—even surpasses the performance of DepQBF with other configurations of constraint learning.

Due to space constraints, several proofs had to be omitted.

2 Preliminaries

Formulas and Assignments. A *literal* is a negated or unnegated variable. If x is a variable, we write $\overline{x} = \neg x$ and $\overline{\neg x} = x$, and let $var(x) = var(\neg x) = x$. If X is a set of literals, we write $\overline{X}$ for the set $\{\, \overline{x} : x \in X \,\}$. A *clause* is a finite disjunction of literals. We call a clause *tautological* if it contains the same variable negated as well as unnegated. A *CNF formula* is a finite conjunction of non-tautological clauses. Whenever convenient, we treat clauses as sets of literals, and CNF formulas as sets of sets of literals. We write $var(C)$ for the set of variables occuring (negated or unnegated) in a clause C, that is, $var(C) = \{\, var(\ell) : \ell \in C \,\}$. Moreover, we let $var(\varphi) = \bigcup_{C \in \varphi} var(C)$ denote the set of variables occurring in a CNF formula φ.

A *truth assignment* (or simply *assignment*) to a set X of variables is a mapping $\tau : X \to \{0, 1\}$. We write $[X]$ for the set of truth assignments to X, and extend $\tau : X \to \{0, 1\}$ to literals by letting $\tau(\neg x) = 1 - \tau(x)$ for $x \in X$. Let $\tau : X \to \{0, 1\}$ be a truth assignment. The restriction $C[\tau]$ of a clause C by τ is defined as follows: if there is a literal $\ell \in C \cap (X \cup \overline{X})$ such that $\tau(\ell) = 1$ then $C[\tau] = 1$. Otherwise, $C[\tau] = C \setminus (X \cup \overline{X})$. The restriction $\varphi[\tau]$ of a CNF formula φ by the assignment τ is defined $\varphi[\tau] = \{\, C[\tau] : C[\tau] \neq 1 \,\}$.

PCNF Formulas. A *PCNF formula* is denoted by $\Phi = \mathcal{Q}.\varphi$, where φ is a CNF formula and $\mathcal{Q} = Q_1 X_1 \dots Q_n X_n$ is a sequence such that $Q_i \in \{\forall, \exists\}$, $Q_i \neq Q_{i+1}$ for $1 \leq i < n$, and the X_i are pairwise disjoint sets of variables. We call φ the *matrix* of Φ and $\mathcal{Q}$ the *(quantifier) prefix* of Φ, and refer to the X_i as *quantifier blocks*. We require that $var(\varphi) = X_1 \cup \dots \cup X_n$ and write $var(\Phi) = var(\varphi)$. We define a partial order $<_\Phi$ on $var(\varphi)$ as $x <_\Phi y \Leftrightarrow x \in X_i, y \in X_j, i < j$. We extend $<_\Phi$ to a relation on literals in the obvious way and drop the subscript whenever Φ is understood. For $x \in var(\Phi)$ we let $R_\Phi(x) = \{\, y \in var(\Phi) : x <_\Phi y \,\}$ and $L_\Phi(x) = \{\, y \in var(\Phi) : y <_\Phi x \,\}$ denote the sets of variables *to the right* and *to the left* of x in Φ, respectively. Relative to the PCNF formula Φ, variable x is called *existential* (*universal*) if $x \in X_i$ and $Q_i = \exists$ ($Q_i = \forall$). The set of existential (universal) variables occurring in Φ is denoted $var_\exists(\Phi)$ ($var_\forall(\Phi)$). The *size* of a PCNF formula $\Phi = \mathcal{Q}.\varphi$ is defined as $|\Phi| = \sum_{C \in \varphi} |C|$. If τ is an assignment, then $\Phi[\tau]$ denotes the PCNF formula $\mathcal{Q}'.\varphi[\tau]$, where $\mathcal{Q}'$ is the

quantifier prefix obtained from $\mathcal{Q}$ by deleting variables that do not occur in $\varphi[\tau]$. *True* and *false* PCNF formulas are defined in the usual way.

Countermodels. Let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula. A *countermodel* of Φ is an indexed family $\{f_u\}_{u \in var_\forall(\Phi)}$ of functions $f_u : [L_\Phi(u)] \to \{0,1\}$ such that $\varphi[\tau] = \{\emptyset\}$ for every assignment $\tau : var(\Phi) \to \{0,1\}$ satisfying $\tau(u) = f_u(\tau|_{L_\Phi(u)})$ for $u \in var_\forall(\Phi)$.

Proposition 1. (Folklore) *A PCNF formula is false if, and only if, it has a countermodel.*

3 Dependency Schemes and LDQ(D)-Resolution

In this section, we introduce the proof system LDQ(D), which combines Q(D)-resolution [36] with long-distance Q-resolution [2]. Q-resolution is a generalization of propositional resolution to PCNF formulas [25]. Q-resolution is of practical interest due to its relation to search based QBF solvers that implement the QDPLL algorithm [12]: the trace of a QDPLL solver generated for a false PCNF formula corresponds to a Q-resolution refutation [17]. QDPLL generalizes the well-known DPLL procedure [14] from SAT to QSAT. In a nutshell, DPLL searches for a satisfying assignment of an input formula by propagating unit clauses and assigning pure literals until the formula cannot be simplified any further, at which point it picks an unassigned variable and branches on the assignment of this variable. Although any of the remaining variables can be chosen for assignment, the order of assignment can have significant effects on the runtime, and modern SAT solvers derived from the DPLL algorithm use sophisticated heuristics to determine what variable to assign next [31].

In QDPLL, the quantifier prefix imposes constraints on the order of variable assignments: a variable may be assigned only if it occurs in the leftmost quantifier block with unassigned variables. Often, this is more restrictive than necessary. For instance, variables from disjoint subformulas may be assigned in any order. Intuitively, a variable can be assigned as long as it *does not depend* on any unassigned variable. This is the intuition underlying a generalization of QDPLL implemented in the solver DepQBF [8,28]. DepQBF uses a *dependency scheme* [34] to compute an overapproximation of variable dependencies. Dependency schemes are mappings that associate every PCNF formula with a binary relation on its variables that refines the order of variables in the quantifier prefix.[2]

Definition 1 (Dependency Scheme). A *dependency scheme* is a mapping D that associates each PCNF formula Φ with a relation $D_\Phi \subseteq \{(x,y) : x <_\Phi y\}$ called the *dependency relation* of Φ with respect to D.

[2] The original definition of dependency schemes [34] is more restrictive than the one given here, but the additional requirements are irrelevant for the purposes of this paper.

The mapping which simply returns the prefix ordering of an input formula can be thought of as a baseline dependency scheme:

Definition 2 (Trivial Dependency Scheme). The *trivial dependency scheme* $\mathrm{D}^{\mathrm{trv}}$ associates each PCNF formula Φ with the relation $\mathrm{D}^{\mathrm{trv}}_{\Phi} = \{ (x, y) : x <_{\Phi} y \}$.

DepQBF uses a dependency relation to determine the order in which variables can be assigned: if y is a variable and there is no unassigned variable x such that (x, y) is in the dependency relation, then y is considered ready for assignment. DepQBF also uses the dependency relation to generalize the $\forall$-reduction rule used in clause learning [8]. As a result of its use of dependency schemes, DepQBF generates proofs in a generalization of Q-resolution called Q(D)-resolution [36], a proof system that takes a dependency scheme D as a parameter.

Dependency schemes can be partially ordered based on their dependency relations: if the dependency relation computed by a dependency scheme D_1 is a subset of the dependency relation computed by a dependency scheme D_2, then D_1 is *more general* than D_2. The more general a dependency scheme, the more freedom DepQBF has in choosing decision variables. Currently, (aside from the trivial dependency scheme) DepQBF supports the so-called *standard dependency scheme* [34].[3] We will work with the more general *reflexive resolution-path dependency scheme* [36], a variant of the resolution-path dependency scheme [35,38]. This dependency scheme computes an overapproximation of variable dependencies based on whether two variables are connected by a (pair of) resolution path(s).

Definition 3 (Resolution Path). Let $\Phi = \mathcal{Q}.\varphi$ be a PCNF formula and let X be a set of variables. A *resolution path* (from ℓ_1 to ℓ_{2k}) via X (in Φ) is a sequence $\ell_1, \ldots, \ell_{2k}$ of literals satisfying the following properties:

1. For all $i \in [k]$, there is a $C_i \in \varphi$ such that $\ell_{2i-1}, \ell_{2i} \in C_i$.
2. For all $i \in [k]$, $var(\ell_{2i-1}) \neq var(\ell_{2i})$.
3. For all $i \in [k-1]$, $\{\ell_{2i}, \ell_{2i+1}\} \subseteq X \cup \overline{X}$.
4. For all $i \in [k-1]$, $\overline{\ell_{2i}} = \ell_{2i+1}$.

If $\pi = \ell_1, \ldots, \ell_{2k}$ is a resolution path in Φ via X, we say that ℓ_1 and ℓ_{2k} are *connected in* Φ (with respect to X). For every $i \in \{1, \ldots, k\}$ we say that π *goes through* $var(\ell_{2i})$.

One can think of a resolution path as a potential chain of implications: if each clause C_i contains exactly two literals, then assigning ℓ_1 to 0 requires setting ℓ_{2k} to 1. If, in addition, there is such a path from $\overline{\ell_1}$ to $\overline{\ell_{2k}}$, then ℓ_1 and ℓ_{2k} have to be assigned the same value. Accordingly, the resolution path dependency scheme identifies variables connected by a pair of resolution paths as potentially dependent on each other.

[3] Strictly speaking, it uses a refined version of the standard dependency scheme [28, p. 49].

$$\frac{}{C}\ \text{(input clause)} \qquad \frac{C_1 \vee e \qquad \neg e \vee C_2}{C_1 \vee C_2}\ \text{(resolution)}$$

An *input clause* $C \in \varphi$ can be used as an axiom. From two clauses $C_1 \vee e$ and $\neg e \vee C_1$, where e is an existential variable, the *(long-distance) resolution* rule can derive the clause $C_1 \vee C_2$, provided that $e < var(\ell)$ for every universal literal ℓ such that $\ell \in C_1$ and $\overline{\ell} \in C_2$.

$$\frac{C}{C \setminus \{u, \neg u\}}\ \text{(}\forall\text{-reduction)}$$

The $\forall$*-reduction* rule derives the clause $C \setminus \{u, \neg u\}$ from C, where $u \in var(C)$ is a universal variable such that $(u, e) \notin D_\Phi$ for every existential variable $e \in var(C)$.

Fig. 1. Derivation rules of LDQ(D)-resolution for a PCNF formula $\Phi = \mathcal{Q}.\varphi$.

Definition 4 (Dependency pair). Let Φ be a PCNF formula and $x, y \in var(\Phi)$. We say $\{x, y\}$ is a *resolution-path dependency pair* of Φ with respect to $X \subseteq var_\exists(\Phi)$ if at least one of the following conditions holds:

- x and y, as well as $\neg x$ and $\neg y$, are connected in Φ with respect to X.
- x and $\neg y$, as well as $\neg x$ and y, are connected in Φ with respect to X.

Definition 5. The *reflexive resolution-path dependency scheme* is the mapping $\mathrm{D}^{\mathrm{rrs}}$ that assigns to each PCNF formula $\Phi = \mathcal{Q}.\varphi$ the relation $\mathrm{D}^{\mathrm{rrs}}_\Phi = \{\, x <_\Phi y : \{x, y\}$ is a resolution-path dependency pair in Φ with respect to $R_\Phi(x) \setminus var_\forall(\Phi)\,\}$.

Both Q-resolution and Q(D)-resolution only allow for the derivation of non-tautological clauses, that is, clauses that do not contain a literal negated as well as unnegated. *Long-distance Q-resolution* is a variant of Q-resolution that admits tautological clauses in certain cases [2]. Variants of QDPLL that allow for learnt clauses to be tautological [39,40] have been shown to generate proofs in long-distance Q-resolution [16].

We combine long-distance resolution with Q(D)-resolution to obtain long-distance Q(D)-resolution (LDQ(D)-resolution). The derivation rules of LDQ(D)-resolution are shown in Fig. 1. Here, as in the rest of the paper, D denotes an arbitrary dependency scheme.

A derivation in a proof system consists of repeated applications of the derivation rules to derive a clause from the clauses of an input formula. Here, derivations will be represented by node-labeled directed acyclic graphs (DAGs). More specifically, we require these DAGs to have a unique sink (that is, a node without outgoing edges) and each of their nodes to have at most two incoming edges. We further assume an ordering on the in-neighbors (or parents) of every node with two incoming edges—that is, each node has a "first" and a "second" in-neighbor. Referring to such DAGs as *proof DAGs*, we define the following two operations to represent resolution and $\forall$-reduction:

1. If ℓ is a literal and $\mathcal{P}_1$ and $\mathcal{P}_2$ are proof DAGs with distinct sinks v_1 and v_2, then $\mathcal{P}_1 \odot_\ell \mathcal{P}_2$ is the proof DAG consisting of the union of $\mathcal{P}_1$ and $\mathcal{P}_2$ along with a new sink v that has two incoming edges, the first one from v_1 and the second one from v_2. Moreover, if C_1 is the label of v_1 in $\mathcal{P}_1$ and C_2 is the label of v_2 in $\mathcal{P}_2$, then v is labeled with the clause $(C_1 \setminus \{\ell\}) \cup (C_2 \setminus \{\bar{\ell}\})$.
2. If u is a variable and $\mathcal{P}$ is a proof DAG with a sink w labeled with C, then $\mathcal{P} - u$ denotes the proof DAG obtained from $\mathcal{P}$ by adding an edge from w to a new node v such that v is labeled with $C \setminus \{u, \neg u\}$.

Definition 6. (Derivation). An *LDQ(D)-resolution derivation* (or *LDQ(D)-derivation*) of a clause C from a PCNF formula $\Phi = \mathcal{Q}.\varphi$ is a proof DAG $\mathcal{P}$ satisfying the following properties. Source nodes are labeled with input clauses from φ. If a node with label C has parents labeled C_1 and C_2 then C can be derived from C_1 and C_2 by (long-distance) resolution. If a node labeled with a clause C has a single parent with label C' then C can be derived from C' by $\forall$-reduction with respect to the dependency scheme D. We refer to these nodes as *input nodes*, *resolution nodes*, and $\forall$ *-reduction nodes*, respectively.

Let $\mathcal{P}$ be an LDQ(D)-derivation from a PCNF formula Φ. The (clause) label of the sink node is called the *conclusion* of $\mathcal{P}$, denoted $Cl(\mathcal{P})$. If the conclusion of $\mathcal{P}$ is the empty clause then we refer to $\mathcal{P}$ as an *LDQ(D)-refutation* of Φ. For a node v of $\mathcal{P}$, the *subderivation* (of $\mathcal{P}$) rooted at v is the proof DAG induced by v and its ancestors in $\mathcal{P}$. It is straightforward to verify that the resulting proof DAG is again an LDQ(D)-derivation from Φ. For convenience, we will identify (sub)derivations with their sinks. The *size* of $\mathcal{P}$, denoted $|\mathcal{P}|$, is the total number of literal occurrences in clause labels of $\mathcal{P}$.

4 Soundness of and Strategy Extraction for LDQ($\mathbf{D^{rrs}}$)

A PCNF formula can be associated with an evaluation game played between an existential and a universal player. These players take turns assigning quantifier blocks in the order of the prefix. The existential player wins if the matrix evaluates to 1 under the resulting variable assignment, while the universal player wins if the matrix evaluates to 0. One can show that the formula is true (false) if and only if the existential (universal) player has a winning strategy in this game, and this winning strategy is a (counter)model.

Goultiaeva, Van Gelder and Bacchus [20] proved that a Q-resolution refutation can be used to compute winning moves for the universal player in the evaluation game. The idea is that universal maintains a "restriction" of the refutation by the assignment constructed in the evaluation game, which is a refutation of the restricted formula.

For assignments made by the existential player, the universal player only needs to consider each instance of resolution whose pivot variable is assigned: one of the premises is not satisifed and can be used to (re)construct a refutation.

When it is universal's turn, the quantifier block for which she needs to pick an assignment is leftmost in the restricted formula. This means that $\forall$-reduction

of these variables is blocked by any of the remaining existential variables and can only be applied to a purely universal clause. In a Q-resolution refutation, these variables must therefore be reduced at the very end, and because Q-resolution does not permit tautological clauses, only one polarity of each universal variable from the leftmost block can appear in a refutation. It follows that universal can maintain a Q-resolution refutation by assigning variables from the leftmost block in such a way as to map the associated literals to 0, effectively deleting them from the remaining Q-resolution refutation.

In this manner, the universal player can maintain a refutation until the end of the game, when all variables have been assigned. At that point, a refutation can consist only of the empty clause, which means that the assignment chosen by the two players falsifies a clause of the original matrix and universal has won the game.

Egly, Lonsing, and Widl [16] observed that this argument goes through even in the case of long-distance Q-resolution, since a clause containing both u and $\neg u$ for a universal variable u can only be derived by resolving on an existential variable to the left of u, but no such existential variable exists if u is from the leftmost block.

In this section, we will prove that this argument can be generalized to LDQ($\mathrm{D}^{\mathrm{rrs}}$)-refutations. We illustrate this correspondence with an example:

Example 1. Consider the PCNF formula

$$\Phi = \exists x\, \forall u\, \exists e, y \quad (x \vee u \vee \overline{y}) \wedge (\overline{x} \vee \overline{u} \vee \overline{y}) \wedge (x \vee y) \wedge (\overline{x} \vee e) \wedge (\overline{u} \vee y) \wedge (\overline{y} \vee e)$$

Figure 2 shows an LDQ($\mathrm{D}^{\mathrm{rrs}}$)-refutation of Φ. The only universal variable is u, so a strategy for the universal player in the evaluation game associated with Φ has to determine an assignment to u given an assignment to x, the only (existential) variable preceding u. The figure illustrates how to compute the assignment to u for the two possible assignments $\tau : \{x\} \to \{0, 1\}$ from the restriction of the refutation by τ. In both cases, only one polarity of u occurs in the restricted refutation and therefore it is easy for universal to determine the correct assignment. Notice that in one of the cases, a generalized $\forall$-reduction node remains present in the restriction—this shows that we cannot limit ourselves to looking at the final reduction step in the proof when looking for the variables to assign (as is the case with ordinary Q-resolution refutations, cf. [20]).

In all of the above cases, the key property that allows universal to maintain a refutation is that universal variables from a leftmost quantifier block may appear in at most one polarity. We will show that, indeed, this property is sufficient for soundness of LDQ(D) when combined with a natural monotonicity property of dependency schemes.

Definition 7. Let D be a dependency scheme. We say that D is *monotone*, if for every PCNF formula Φ and every assignment τ to a subset of var_Φ, $D_{\Phi[\tau]} \subseteq D_\Phi$. We say that D is *simple*, if for every PCNF formula $\Phi = \forall X\, \mathcal{Q}.\varphi$, every LDQ(D)-derivation $\mathcal{P}$ from Φ, and every universal variable $u \in X$, u or $\overline{u}$ does not appear in $\mathcal{P}$. We say that D is *normal* if it is both monotone and simple.

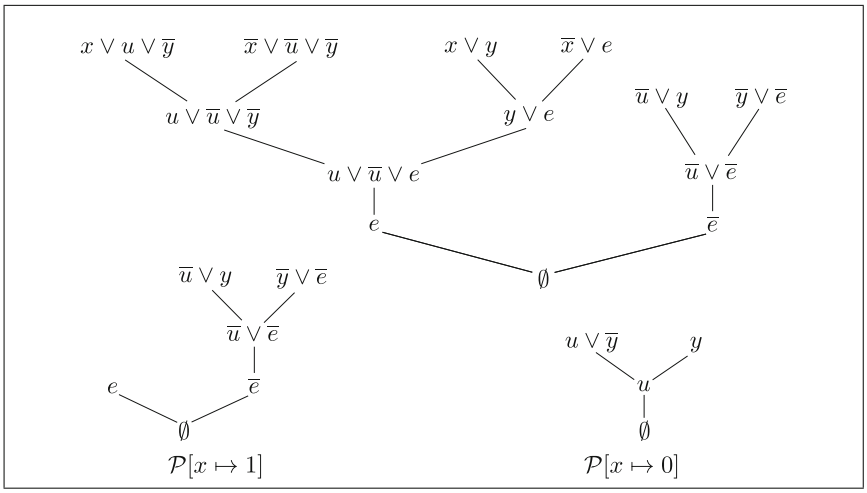

Fig. 2. An LDQ($\mathrm{D}^{\mathrm{rrs}}$)-refutation of the formula Φ from Example 1 and two restrictions.

As in the case of Q-resolution, universal's move for a particular quantifier block can be computed from the assignment corresponding to the previous moves and the refutation in polynomial time. Since every polynomial-time algorithm can be implemented by a family of polynomially-sized circuits, and because these circuits can even be computed in polynomial time [1, p. 109], it follows that LDQ(D) admits polynomial-time strategy extraction when D is normal. While the strategy extraction algorithm based on these general considerations is unlikely to be efficient, the algorithm for computing winning moves for universal is simple enough to be amenable to efficient simulation by a Boolean circuit. In Sect. 4.1, we give a direct construction that leads to the following result.

Theorem 1. *Let D be a normal dependency scheme. Then, there is an algorithm that computes a countermodel of a PCNF formula Φ with n variables from an LDQ(D)-refutation $\mathcal{P}$ of Φ in time $O(|\mathcal{P}| \cdot n)$.*

As an application of this general result, we will prove that the reflexive resolution-path dependency scheme is normal in Sect. 4.2.

Theorem 2. *D^{rrs} is normal.*

Corollary 1. *There is an algorithm that computes a countermodel of a PCNF formula Φ with n variables from an LDQ(D^{rrs})-refutation $\mathcal{P}$ of Φ in time $O(|\mathcal{P}| \cdot n)$.*

This result immediately carries over to the less general standard dependency scheme:

Corollary 2. *There is an algorithm that computes a countermodel of a PCNF formula Φ with n variables from an LDQ(D^{std})-refutation $\mathcal{P}$ of Φ in time $O(|\mathcal{P}| \cdot n)$.*

In combination with Proposition 1, these results imply soundness of both proof systems.

Corollary 3. *The systems LDQ(D^{std}) and LDQ(D^{rrs}) are sound.*

4.1 Certificate Extraction for Normal Dependency Schemes

We begin by formally defining the "restriction" of an LDQ(D)-derivation by an assignment, which is a straightforward generalization of this operation for Q-resolution derivations [20].[4] The result of restricting a derivation is either a derivation or the object $\top$, which can be interpreted as representing the tautological clause containing every literal. Accordingly, we stipulate that $\ell \in \top$ for every literal ℓ.

Definition 8. (Restriction). Let Φ be a PCNF formula and let $\mathcal{P}$ be an LDQ(D)-derivation from Φ. Further, let $X \subseteq var(\Phi)$ and let $\tau : X \to \{0,1\}$ be a truth assignment. The *restriction of* $\mathcal{P}$ by τ, in symbols $\mathcal{P}[\tau]$, is defined as follows.

1. If $\mathcal{P}$ is an input node there are two cases. If $Cl(\mathcal{P})[\tau] = 1$ then $\mathcal{P}[\tau] = \top$. Otherwise, $\mathcal{P}[\tau]$ is the proof DAG consisting of a single node labeled with $Cl(\mathcal{P})[\tau]$.
2. If $\mathcal{P} = \mathcal{P}_1 \odot_\ell \mathcal{P}_2$, that is, if $\mathcal{P}$ is a resolution node, we distinguish four cases:
   (a) If $\ell \notin Cl(\mathcal{P}_1[\tau])$ then $\mathcal{P}[\tau] = \mathcal{P}_1[\tau]$.
   (b) If $\ell \in Cl(\mathcal{P}_1[\tau])$ and $\overline{\ell} \notin Cl(\mathcal{P}_2[\tau])$ then $\mathcal{P}[\tau] = \mathcal{P}_2[\tau]$.
   (c) If $\ell \in Cl(\mathcal{P}_1[\tau])$, $\overline{\ell} \in Cl(\mathcal{P}_2[\tau])$, and $\mathcal{P}_1[\tau] = \top$ or $\mathcal{P}_2[\tau] = \top$, we let $\mathcal{P}[\tau] = \top$.
   (d) If $\ell \in Cl(\mathcal{P}_1[\tau])$, $\overline{\ell} \in Cl(\mathcal{P}_2[\tau])$, $\mathcal{P}_1[\tau] \neq \top$, and $\mathcal{P}_2[\tau] \neq \top$, we define $\mathcal{P}[\tau] = \mathcal{P}_1[\tau] \odot_\ell \mathcal{P}_2[\tau]$.
3. If $\mathcal{P} = \mathcal{P}' - u$, that is, if $\mathcal{P}$ is a $\forall$-reduction node, we distinguish three cases:
   (a) If $\mathcal{P}'[\tau] = \top$ then $\mathcal{P}[\tau] = \top$.
   (b) If $\mathcal{P}'[\tau] \neq \top$ and $u \notin var(Cl(\mathcal{P}'[\tau]))$ then $\mathcal{P}[\tau] = \mathcal{P}'[\tau]$.
   (c) If $\mathcal{P}'[\tau] \neq \top$ and $u \in var(Cl(\mathcal{P}'[\tau]))$ then $\mathcal{P}[\tau] = \mathcal{P}'[\tau] - u$.

If D is a monotone dependency scheme, LDQ(D)-refutations are preserved under restriction by an existential assignment (cf. [20, Lemma4]). This is stated in the following lemma, which can by proved by a straightforward induction on the structure of the LDQ(D)-derivation.

Lemma 1. *Let D be a monotone dependency scheme, let $\mathcal{P}$ be an LDQ(D)-derivation from a PCNF formula Φ, let $E \subseteq var_\exists(\Phi)$, and let $\tau : E \to \{0,1\}$ be an assignment. If $\mathcal{P}[\tau] = \top$ then $Cl(\mathcal{P})[\tau] = 1$. Otherwise, $\mathcal{P}[\tau]$ is an LDQ(D)-derivation from $\Phi[\tau]$ such that $Cl(\mathcal{P}[\tau]) \subseteq Cl(\mathcal{P})[\tau]$.*

[4] Our definition slightly differs from the original for the resolution rule: if restriction removes the pivot variable from both premises, we simply pick the (restriction of the) first premise as the result (rather than the clause containing fewer literals). This simplifies the certificate extraction argument given below.

Above, we argued that the universal player can use an LDQ(D)-refutation for a normal dependency scheme D in order to compute winning moves in the evaluation game associated with a PCNF formula and that this can be used to compute a countermodel of the formula in polynomial time. We now prove this directly, by showing how to construct a circuit implementing a countermodel from an LDQ(D)-refutation.

We begin by describing auxiliary circuits simulating the restriction operation. Let $\Phi = Q_1 X_1 \dots Q_k X_k . \varphi$ be a PCNF formula and let $\mathcal{P}$ be a refutation of Φ. For each quantifier block X_i, each subderivation $\mathcal{S}$ of $\mathcal{P}$, and each literal ℓ, we will construct circuits $\textsc{top}^i_{\mathcal{S}}$ and $\textsc{contains}^i_{\mathcal{S},\ell}$ with inputs from $X = \bigcup_{j<i} X_j$ such that, for every assignment $\sigma : X \to \{0,1\}$,

$$\textsc{top}^i_{\mathcal{S}}[\sigma] = 1 \iff \mathcal{S}[\sigma] = \top \tag{1}$$

$$\textsc{contains}^i_{\mathcal{S},\ell}[\sigma] = 1 \iff \ell \in Cl(\mathcal{S}[\sigma]) \tag{2}$$

We first describe our construction and then prove that it satisfies the above properties in Lemma 2. Let $\mathcal{S}$ be an input node. We let

$$\textsc{top}^1_{\mathcal{S}} := \bigvee \{\, Cl(\mathcal{S}) \cap (X_1 \cup \overline{X_1}) \,\},$$

and define $\textsc{top}^i_{\mathcal{S}}$ for $1 < i \leq k$ as

$$\textsc{top}^i_{\mathcal{S}} := \textsc{top}^{i-1}_{\mathcal{S}} \vee \bigvee \{\, Cl(\mathcal{S}) \cap (X_i \cup \overline{X_i}) \,\}.$$

Moreover, for $1 \leq i \leq k$ we define $\textsc{contains}^i_{\mathcal{S},\ell}$ as

$$\textsc{contains}^i_{\mathcal{S},\ell} = \begin{cases} 1 & \text{if } \ell \in Cl(\mathcal{S}) \setminus (X \cup \overline{X}), \\ \textsc{top}^i_{\mathcal{S}} & \text{otherwise.} \end{cases}$$

For non-input nodes, we proceed as follows. If $\mathcal{S} = \mathcal{S}_1 \odot_q \mathcal{S}_2$, we define $\textsc{top}^i_{\mathcal{S}}$ as

$$\textsc{top}^i_{\mathcal{S}} = (\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \textsc{top}^i_{\mathcal{S}_2}) \vee (\textsc{contains}^i_{\mathcal{S}_2,\overline{q}} \wedge \textsc{top}^i_{\mathcal{S}_1}),$$

and if $\mathcal{S} = \mathcal{S}' - u$, we let

$$\textsc{top}^i_{\mathcal{S}} := \textsc{top}^i_{\mathcal{S}'}.$$

For the $\textsc{contains}^i_{\mathcal{S},\ell}$ circuit, we distinguish two cases. Let ℓ be a literal and $\mathcal{S}$ a derivation. If $\ell \notin Cl(\mathcal{S})$ we simply let

$$\textsc{contains}^i_{\mathcal{S},\ell} := \textsc{top}^i_{\mathcal{S}}.$$

Otherwise, if $\ell \in Cl(\mathcal{S})$, we have to consider two cases. First, if $\mathcal{S} = \mathcal{S}_1 \odot_q \mathcal{S}_2$, we let

$$\begin{aligned}\textsc{contains}^i_{\mathcal{S},\ell} = &\textsc{top}^i_{\mathcal{S}} \vee \\ &(\neg\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \textsc{contains}^i_{\mathcal{S}_1,\ell}) \vee \\ &(\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \neg\textsc{contains}^i_{\mathcal{S}_2,\overline{q}} \wedge \textsc{contains}^i_{\mathcal{S}_2,\ell}) \vee \\ &(\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \textsc{contains}^i_{\mathcal{S}_2,\overline{q}} \wedge (\textsc{contains}^i_{\mathcal{S}_1,\ell} \vee \textsc{contains}^i_{\mathcal{S}_2,\ell})).\end{aligned}$$

Second, if $\mathcal{S} = \mathcal{S}' - u$, then

$$\textsc{contains}^i_{\mathcal{S},\ell} := \textsc{contains}^i_{\mathcal{S}',\ell}.$$

To implement the winning strategy for universal sketched above, we further construct circuits $\textsc{polarity}_{\mathcal{S},u}$ for each node $\mathcal{S}$ of $\mathcal{P}$ and each universal variable $u \in var_\forall(\Phi)$, such that, for each assignment $\tau : L_\Phi(u) \to \{0,1\}$,

$$\textsc{polarity}_{\mathcal{S},u}[\tau] = 1 \iff u \text{ occurs in } \mathcal{S}[\tau]. \tag{3}$$

Let $u \in X_i$ be a universal variable from the ith quantifier block. If $\mathcal{S}$ is an input node, we simply define

$$\textsc{polarity}_{\mathcal{S},u} := \textsc{contains}^i_{\mathcal{S},u},$$

and if $\mathcal{S} = \mathcal{S}' - u$ is a $\forall$-reduction node, we let

$$\textsc{polarity}_{\mathcal{S},u} := \textsc{polarity}_{\mathcal{S}',u}.$$

If $\mathcal{S} = \mathcal{S}_1 \odot_q \mathcal{S}_2$, then

$$\begin{aligned}\textsc{polarity}_{\mathcal{S},u} := &(\neg\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \textsc{polarity}_{\mathcal{S}_1,u}) \vee \\ &(\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \neg\textsc{contains}^i_{\mathcal{S}_2,\overline{q}} \wedge \textsc{polarity}_{\mathcal{S}_2,u}) \vee \\ &(\textsc{contains}^i_{\mathcal{S}_1,q} \wedge \textsc{contains}^i_{\mathcal{S}_2,\overline{q}} \wedge \\ &\qquad (\textsc{polarity}_{\mathcal{S}_1,u} \vee \textsc{polarity}_{\mathcal{S}_2,u})).\end{aligned}$$

Lemma 2. *Let $\Phi = Q_1X_1 \ldots Q_kX_k.\varphi$ be a PCNF formula and let $\mathcal{P}$ be an LDQ(D)-derivation from Φ. For each $1 \leq i \leq k$, each literal ℓ, every $u \in var_\forall(\Phi) \cap X_i$, and every truth assignment $\sigma : \bigcup_{j=1}^{i-1} X_j \to \{0,1\}$, $\textsc{top}^i_{\mathcal{P}}$ satisfies (1), $\textsc{contains}^i_{\mathcal{P},\ell}$ satisfies (2), and $\textsc{polarity}_{\mathcal{P},u}$ satisfies (3).*

These auxiliary circuits can be efficiently constructed in a top-down manner, from the input nodes to the conclusion. By a careful analysis, we obtain the following:

Lemma 3. *There is an algorithm that, given a PCNF formula Φ and an LDQ(D)-derivation $\mathcal{P}$ from Φ, computes the circuits $\textsc{polarity}_{\mathcal{P},u}$ for every universal variable u in time $O(|\mathcal{P}| \cdot n)$, where $n = |var(\Phi)|$.*

Using Lemma 1, we can spell out the argument sketched at the beginning of this section and prove that for normal dependency schemes D, the universal player can maintain an LDQ(D)-refutation throughout the evaluation game by successively restricting an initial LDQ(D)-refutation by both players' moves and assigning universal variables from the leftmost remaining block X so as to falsify the (unique) literals from X remaining the refutation. Lemma 2 tells us that the POLARITY circuits can be used to implement this strategy, which leads to the following lemma.

Lemma 4. *Let D be a normal dependency scheme, let $\mathcal{P}$ be an LDQ(D)-refutation of a PCNF formula Φ. Then the family $\{f_u\}_{u \in var_\forall(\Phi)}$ of functions $f_u = \neg\text{POLARITY}_{\mathcal{P},u}$ is a countermodel of Φ.*

Theorem 1 immediately follows from from Lemmas 3 and 4.

4.2 The Reflexive Resolution-Path Dependency Scheme is Normal

In order to prove Theorem 2 and show that D^{rrs} is normal, we will need some insight into the relationship between resolution paths and LDQ(D)-derivations. We begin by observing that no clause derived by LDQ(D) from a PCNF formula $\forall X \mathcal{Q}.\varphi$ can contain both u and $\neg u$ if $u \in X$, as such a clause would have to be derived by resolving on a variable to the left of u.

Lemma 5. *Let $\mathcal{P}$ be an LDQ(D)-derivation from a PCNF formula $\Phi = \forall X \mathcal{Q}.\varphi$ and let $u \in X$. Then $Cl(\mathcal{P})$ cannot contain both u and $\neg u$.*

Let $\mathcal{P}$ be an LDQ(D)-refutation of a PCNF formula Φ and let u be a universal variable such that both u and $\neg u$ appear in $\mathcal{P}$. Since $\mathcal{P}$ is a refutation, both u and $\neg u$ have to be eventually removed by $\forall$-reduction. By the preceding lemma, u and $\neg u$ cannot occur together in a clause. Now, if C is a purely universal clause appearing in $\mathcal{P}$, then C must be followed by a sequence of $\forall$-reduction steps that lead to the empty clause. Since $\mathcal{P}$ contains only a single node labeled with the empty clause, this means that each purely universal clause appearing in $\mathcal{P}$ must be part of a single final sequence of $\forall$-reduction steps concluding the refutation, so that $C \subseteq C'$ or $C' \subseteq C$ holds for every pair C, C' of such clauses. It follows that we cannot have a purely universal clause C with $u \in C$ and a purely universal clause C' with $\neg u \in C'$ such that C and C' both occur in $\mathcal{P}$. So at least one of u and $\neg u$ must be reduced from a clause which still contains an existential variable. We will prove that $(u, e) \in \text{D}^{\text{rrs}}_\Phi$ for some such variable e.

To show this, we establish a connection between resolution paths and the structure of LDQ(D)-refutations: resolution paths can be used to determine whether two literals may occur together in a clause derived by LDQ(D). The following statement can be proved by a simple induction on the structure of $\mathcal{P}$ (cf. [35,38]).

Lemma 6. *Let $\mathcal{P}$ be an LDQ(D)-derivation from a PCNF formula Φ and let ℓ, ℓ' be literals in $Cl(\mathcal{P})$. Then ℓ and ℓ' are connected in Φ with respect to the set of variables appearing as pivots in $\mathcal{P}$.*

If a clause C' appears in the derivation of a clause C and $C \not\subseteq C'$, then C cannot be derived from C' by $\forall$-reduction alone, and some existential variable $e \in var(C')$ must be resolved on in the derivation of C. It follows that there have to be resolution paths connecting literals in C' with literals in $C \setminus C'$ that go through an existential variable appearing in C'. This is made precise in the following lemma.

Lemma 7. *Let $\mathcal{P}$ be an LDQ(D)-derivation from a PCNF formula Φ, let $\mathcal{P}'$ be a node of $\mathcal{P}$, and let $\ell' \in Cl(\mathcal{P}')$. For each literal $\ell \in Cl(\mathcal{P}) \setminus Cl(\mathcal{P}')$, there is an existential variable $e \in var(Cl(\mathcal{P}'))$ and a resolution path from ℓ' to ℓ in Φ via the set of pivots in $\mathcal{P}$ that goes through e.*

Together, the two preceding lemmas can be used to show that an LDQ(D)-refutation that contains both u and $\neg u$ from an outermost universal quantifier block induces a resolution path from u to $\neg u$ through an existential variable e such that u and e or $\neg u$ and e occur together in a clause.

Lemma 8. *Let $\mathcal{P}$ be an LDQ(D)-refutation of a PCNF formula $\Phi = \forall X \mathcal{Q}.\varphi$ and let $u \in X$ be a universal variable such that $\mathcal{P}$ contains nodes $\mathcal{P}_1$ and $\mathcal{P}_2$ with $u \in Cl(\mathcal{P}_1)$ and $\neg u \in Cl(\mathcal{P}_2)$. Then there is an existential variable $e \in var(Cl(P_1) \cup Cl(P_2))$ and a resolution path in Φ from u to $\neg u$ via variables resolved on in $\mathcal{P}$ that goes through e.*

Proof (of Theorem 2). We first prove that $\mathrm{D}^{\mathrm{rrs}}$ is simple. Let $\mathcal{P}$ be an LDQ($\mathrm{D}^{\mathrm{rrs}}$)-refutation of a PCNF formula $\Phi = \forall X \mathcal{Q}.\varphi$ and let $u \in X$. Suppose that u and $\neg u$ both appear in $\mathcal{P}$. Since $\mathcal{P}$ is a refutation, there have to be $\forall$-reduction nodes $\mathcal{P}_1 = \mathcal{P}'_1 - u$ such that $u \in Cl(\mathcal{P}'_1)$ and $\mathcal{P}_2 = \mathcal{P}'_2 - u$ such that $\neg u \in Cl(\mathcal{P}'_2)$. By Lemma 8, there has to be a resolution path π from u to $\neg u$ via variables resolved on in $\mathcal{P}$ such that π goes through an existential variable $e \in var(Cl(P_1) \cup Cl(P_2))$. We can think of π as the concatentation of two resolution paths π_1 and π_2, where π_1 is a resolution path from u to ℓ and π_2 is a resolution path from $\overline{\ell}$ to $\neg u$, where ℓ is a literal with $var(\ell) = e$. That is, $\{u, e\}$ is a resolution-path dependency pair of Φ with respect to variables resolved on in $\mathcal{P}$. Since u is leftmost in the quantifier prefix, $\mathcal{P}$ only resolves on existential variables to the right of u and thus $(u, e) \in \mathrm{D}^{\mathrm{rrs}}_{\Phi}$. But $(u, e) \notin \mathrm{D}^{\mathrm{rrs}}_{\Phi}$ for each existential variable $e \in var(Cl(P'_1) \cup Cl(P'_2))$ by definition of LDQ($\mathrm{D}^{\mathrm{rrs}}$), a contradiction. To see that $\mathrm{D}^{\mathrm{rrs}}$ is monotone, note that a resolution path in any restriction $\Phi[\tau]$ of a PCNF formula Φ is also a resolution path in Φ. □

5 Experiments

To gauge the potential of clause learning based on LDQ($\mathrm{D}^{\mathrm{std}}$), we ran experiments with the search-based solver DepQBF.[5] By default, DepQBF supports proof generation only in combination with the trivial dependency scheme—in

[5] http://lonsing.github.io/depqbf/

that case, it generates Q-resolution or long distance Q-resolution proofs (depending on whether long distance resolution is enabled). However, by uncommenting a few lines in the source code, proof generation can also be enabled with the standard dependency scheme, and this option can even be combined with long distance resolution. For false formulas, the resulting proofs are Q(D^{std})-resolution or LDQ(D^{std})-resolution refutations.

We compared the performance of DepQBF in four configurations,[6] each using a different proof system for constraint learning:

1. Long distance Q-resolution with ∀/∃-reduction according to D^{std} (LDQD).
2. Long distance Q-resolution with ordinary ∀/∃-reduction (LDQ).
3. Q-resolution with ∀/∃-reduction according to D^{std} (QD).
4. Ordinary Q-resolution (Q).

These experiments were performed on a cluster with Intel Xeon E5649 processors at 2.53 GHz running 64-bit Linux. We set time and memory limits of 900 seconds and 4 GB, respectively.

Instances were taken from two tracks of the QBF Gallery 2014: the *applications* track consisting of 6 instance families and a total of 735 formulas, and the *QBFLib* track consisting of 276 formulas.

For our first set of experiments, we disabled dynamic QBCE (Quantified Blocked Clause Elimination), a technique introduced with version 5.0 of DepQBF [29]. We further used bloqqer[7] with default settings as a preprocessor. Since LDQ(D^{std}) generalizes both long distance Q-resolution and Q(D^{std})-resolution, we were expecting a performance increase with LDQ(D^{std})-learning compared to learning based on the other proof systems. However, all four configurations showed virtually identical performance on both the application and QBFlib benchmark sets in terms of total runtime and instances solved within the time limit.

To get a more detailed picture, we broke down the results for the application track by instance family, limiting ourselves to instances that were solved by at least one configuration. The barplot in Fig. 3 shows that there are considerable differences in performance between solver configurations for individual instances families, with each solver configuration being outperformed by another configuration on at least one family.

For our second set of experiments, we turned on dynamic QBCE. This led to a significant performance increase both in terms of number of instances solved within the time limit and total runtime for both benchmark sets, a result that is consistent with the findings in [29]. However, as far as the performance of LDQ(D^{std})-learning is concerned, the application and QBFlib tracks differed significantly for this experiment.

While LDQ(D^{std})-learning fared *worst* among the configurations both with respect to instances solved and total runtime on the application track, it was

[6] As a sanity check, we verified that all configurations that were able to solve a particular instance returned the same result.

[7] http://fmv.jku.at/bloqqer/

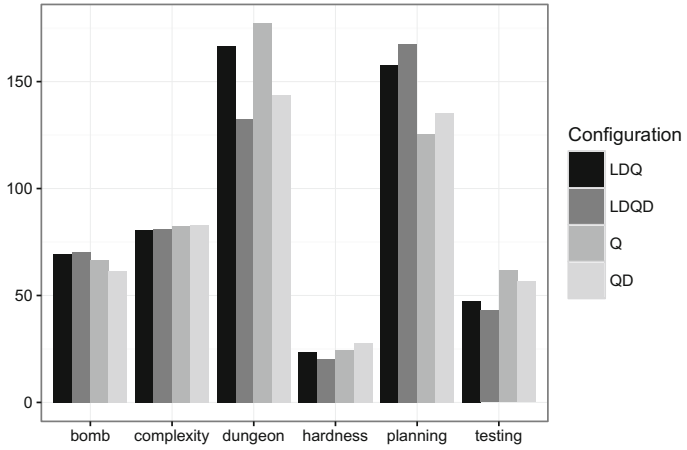

Fig. 3. Average runtime in seconds (y-axis) for instances from the application track for each instance family (x-axis), by solver configuration (with preprocessing, but without dynamic QBCE). Here, we only considered instances that were solved by at least one configuration.

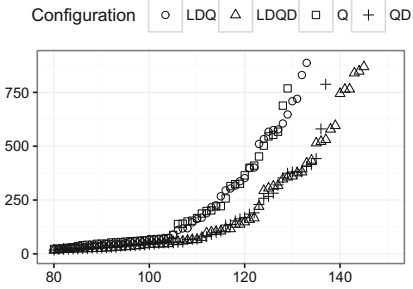

Fig. 4. Solved instances from the QBFLib track (x-axis) sorted by runtime (y-axis), by solver configuration (with preprocessing and dynamic QBCE).

Table 1. Number of solved instances, solved true instances, solved false instances, and total runtime in seconds (including timeouts) for the application track, with QBCE (but without preprocessing)

Configuration	Solved	True	False	Time
LDQD	440	223	217	287012
LDQ	435	223	212	291574
QD	440	225	215	291661
Q	437	221	216	337141

the *best* configuration for the QBFlib track in both respects. As Fig. 4 shows, using the standard dependency scheme was beneficial both with and without long distance resolution for the QBFlib instances.

For our final set of experiments, we left dynamic QBCE enabled but *disabled* preprocessing for the application track, as this was shown to lead to a performance *increase* in the case of learning with ordinary Q-resolution [29]. As expected, this resulted in a performance increase across the board (see Table 1). Moreover, LDQ(D^{std})-learning was the best configuration in terms of instances solved (on par with Q(D^{std})-resolution) as well as in terms of overall runtime.

6 Discussion

The experiments in Sect. 5 show that DepQBF can benefit from learning based on LDQ($\mathrm{D}^{\mathrm{std}}$). This benefit is essentially "for free", in that it does not require any changes to the implementation, but soundness of the resulting solver configuration is not immediate. The results of Sect. 4 contribute to a soundness proof, but they remain partial in two respects: first, soundness of LDQ($\mathrm{D}^{\mathrm{std}}$) only implies that we can trust the solver when it outputs "false". To prove that "true" answers can be trusted as well, one has to show soundness of quantified term resolution when combined with the standard dependency scheme and long distance resolution. Alternatively, one could use LDQ($\mathrm{D}^{\mathrm{std}}$) for clause learning only, in combination with ordinary long distance Q-resolution for cube learning. Second, we observed synergies only when dynamic QBCE was activated, and it remains to show that clause learning based on LDQ($\mathrm{D}^{\mathrm{std}}$) is sound in combination with this technique.

We take Theorem 1 as proof that, in principle, efficient certificate extraction from LDQ($\mathrm{D}^{\mathrm{rrs}}$)-refutations is possible. For practical purposes, the time bound of $O(|\mathcal{P}| \cdot n)$ is not good enough. For LDQ($\mathrm{D}^{\mathrm{std}}$), a modified extraction algorithm achieves a runtime of $O(|\mathcal{P}| \cdot k)$, where k is the number of quantifier alternations of the input formula. A proof-of-concept implementation currently does not scale to proofs larger than a few megabytes, but we are confident that further improvements will lead to an efficient enough algorithm for practical use.

Acknowledgments. We would like to thank Florian Lonsing for helpful discussions and for pointing out how to modify DepQBF so that it generates LDQ($\mathrm{D}^{\mathrm{std}}$) proofs.

References

1. Arora, S., Barak, B.: Computational Complexity - A Modern Approach. Cambridge University Press, New York (2009)
2. Balabanov, V., Jiang, J.R.: Unified QBF certification and its applications. Formal Methods Syst. Des. **41**(1), 45–65 (2012)
3. Balabanov, V., Jiang, J. R., Janota, M., Widl, M.: Efficient extraction of QBF (counter)models from long-distance resolution proofs. In: Bonet, B., Koenig, S. (eds.), Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25–30, Austin, Texas, USA, pp. 3694–3701. AAAI Press (2015)
4. Balabanov, V., Widl, M., Jiang, J.H.R.: QBF resolution systems and their proof complexities. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 154–169. Springer, Heidelberg (2014)
5. Benedetti, M., Mangassarian, H.: QBF-based formal verification: Experience and perspectives. J. Satisfiability, Boolean Model. Comput. **5**(1–4), 133–191 (2008)
6. Beyersdorff, O., Chew, L., Janota, M.: Proof complexity of resolution-based QBF calculi. In: Mayr, E.W., Ollinger, N. (ed.), 32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4–7, 2015, Garching, Germany, vol. 30 of LIPIcs, pp. 76–89. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2015)

7. Beyersdorff, O., Chew, L., Mahajan, M., Shukla, A.: Feasible interpolation for QBF resolution calculi. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9134, pp. 180–192. Springer, Heidelberg (2015)
8. Biere, A., Lonsing, F.: Integrating dependency schemes in search-based QBF solvers. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 158–171. Springer, Heidelberg (2010)
9. Biere, A., Lonsing, F., Seidl, M.: Blocked clause elimination for QBF. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS, vol. 6803, pp. 101–115. Springer, Heidelberg (2011)
10. Bloem, R., Könighofer, R., Seidl, M.: SAT-based synthesis methods for safety specs. In: McMillan, K.L., Rival, X. (eds.) VMCAI 2014. LNCS, vol. 8318, pp. 1–20. Springer, Heidelberg (2014)
11. Bubeck, U.: Model-based transformations for quantified Boolean formulas. Ph.D. thesis, University of Paderborn (2010)
12. Cadoli, M., Schaerf, M., Giovanardi, A., Giovanardi, M.: An algorithm to evaluate Quantified Boolean Formulae and its experimental evaluation. J. Autom. Reasoning **28**(2), 101–142 (2002)
13. Cashmore, M., Fox, M., Giunchiglia, E.: Partially grounded planning as Quantified Boolean Formula. In: Borrajo, D., Kambhampati, S., Oddi, A., Fratini, S. (eds.), 23rd International Conference on Automated Planning and Scheduling, ICApPS. AAAI (2013)
14. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM **5**, 394–397 (1962)
15. Egly, U.: On sequent systems and resolution for QBFs. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 100–113. Springer, Heidelberg (2012)
16. Egly, U., Lonsing, F., Widl, M.: Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR-19 2013. LNCS, vol. 8312, pp. 291–308. Springer, Heidelberg (2013)
17. Giunchiglia, E., Narizzano, M., Tacchella, A.: Clause/term resolution and learning in the evaluation of Quantified Boolean Formulas. J. Artif. Intell. Res. **26**, 371–416 (2006)
18. Goultiaeva, A., Bacchus, F.: Exploiting QBF duality on a circuit representation. In: Fox, M., Poole, D. (eds.), Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI . AAAI Press (2010)
19. Goultiaeva, A., Seidl, M., Biere, A.: Bridging the gap between dual propagation and CNF-based QBF solving. In: Macii, E. (ed.) Design. Automation and Test in Europe. EDA Consortium San Jose, pp. 811–814. ACM DL, CA, USA (2013)
20. Goultiaeva, A., Van Gelder, A., Bacchus, F.: A uniform approach for generating proofs and strategies for both true and false QBF formulas. In Walsh, T. (ed), Proceedings of IJCAI, pp. 546–553. IJCAI/AAAI (2011)
21. Heule, M., Seidl, M., Biere, A.: A unified proof system for QBF preprocessing. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS, vol. 8562, pp. 91–106. Springer, Heidelberg (2014)
22. Janota, M., Chew, L., Beyersdorff, O.: On unification of QBF resolution-based calculi. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part II. LNCS, vol. 8635, pp. 81–93. Springer, Heidelberg (2014)
23. Janota, M., Klieber, W., Marques-Silva, J., Clarke, E.: Solving QBF with counterexample guided refinement. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 114–128. Springer, Heidelberg (2012)

24. Janota, M., Marques-Silva, J.: On propositional QBF expansions and Q-Resolution. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 67–82. Springer, Heidelberg (2013)
25. Kleine Büning, H., Karpinski, M., Flögel, A.: Resolution for quantified Boolean formulas. Inf. Comput. **117**(1), 12–18 (1995)
26. Klieber, W., Sapra, S., Gao, S., Clarke, E.: A non-prenex, non-clausal QBF solver with game-state learning. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 128–142. Springer, Heidelberg (2010)
27. Kronegger, M., Pfandler, A., Pichler, R.: Conformant planning as benchmark for QBF-solvers. In: International Workshop on Quantified Boolean Formulas - QBF (2013). http://fmv.jku.at/qbf2013/
28. Lonsing, F.: Dependency Schemes and Search-Based QBF : Theory and Practice. Ph.D. thesis, Johannes Kepler University, Linz, Austria, Apr 2012
29. Lonsing, F., Bacchus, F., Biere, A., Egly, U., Seidl, M.: Enhancing search-based QBF solving by dynamic blocked clause elimination. In: Davis, M. (ed.) LPAR-20 2015. LNCS, vol. 9450, pp. 418–433. Springer, Heidelberg (2015). doi:10.1007/978-3-662-48899-7_29
30. Lonsing, F., Egly, U., Van Gelder, A.: Efficient clause learning for quantified boolean formulas via QBF pseudo unit propagation. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 100–115. Springer, Heidelberg (2013)
31. Marques-Silva, J.: The impact of branching heuristics in propositional satisfiability algorithms. In: Barahona, P., Alferes, J.J. (eds.) EPIA 1999. LNCS (LNAI), vol. 1695, pp. 62–74. Springer, Heidelberg (1999)
32. Niemetz, A., Preiner, M., Lonsing, F., Seidl, M., Biere, A.: Resolution-based certificate extraction for QBF. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 430–435. Springer, Heidelberg (2012)
33. Rintanen, J.: Asymptotically optimal encodings of conformant planning in QBF. In: 22nd AAAI Conference on Artificial Intelligence, pp. 1045–1050. AAAI (2007)
34. Samer, M., Szeider, S.: Backdoor sets of quantified Boolean formulas. J. Autom. Reasoning **42**(1), 77–97 (2009)
35. Slivovsky, F., Szeider, S.: Computing resolution-path dependencies in linear time. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 58–71. Springer, Heidelberg (2012)
36. Slivovsky, F., Szeider, S.: Soundness of Q-resolution with dependency schemes. Theor. Comput. Sci. **612**, 83–101 (2016)
37. Staber, S., Bloem, R.: Fault localization and correction with QBF. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 355–368. Springer, Heidelberg (2007)
38. Van Gelder, A.: Variable independence and resolution paths for quantified boolean formulas. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 789–803. Springer, Heidelberg (2011)
39. Zhang, L., Malik, S.: Conflict driven learning in a quantified Boolean satisfiability solver. In: Pileggi, L.T. Kuehlmann, A. (eds.), Proceedings of the IEEE/ACM International Conference on Computer-aided Design, ICCAD, San Jose, California, USA, November 10–14, pp. 442–449. ACM/IEEE Computer Society (2002)
40. Zhang, L., Malik, S.: Towards a symmetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 200–215. Springer, Heidelberg (2002)